JS

JS

Roland®

Roland
Rhythm Composer TR-808
Computer Controlled

Roland
TR-909
RHYTHM COMPOSER

DIFFERENT DRUMMER

We don't call the TR-909 a drum machine for some very good reasons. True, it's a machine that makes drum sounds, but that's the end of any similarities between run-of-the-mill drum machines and the TR-909 Rhythm Composer. In fact, playing with the TR-909 is more like playing with a real drummer than anything else. Here's why. The Best Sounds. We start with digital recordings of real drums, then through a 3-D waveform analysis, re-create the sounds through a hybird digital/analog process. Not only does this provide the best drum sounds, but also the most flexible. Change the snap of the snare, the decay of the bass, you call it. The sounds you get are the sounds you really want. Even better—in addition to the 11 internal drum sounds, add up to 16 more drum sounds (digital and analog) through external sound modules. That means 27 drum sounds with no major surgery. The Best Programming. Program a roll on most drum machines and you'll see why they're called machines. That's why the TR-909 gives you the choice of Step Programming (highly visual and accurate) PLUS the additional spontaneity of Real-time Programming. The TR-909 also gives the most expressive and easily programmed dynamics. The Most Flexibility. Think of any way to interface, and you'll find it on the TR-909. MIDI, Sync-24, Tape Memory Save/Load, RAM-Pak Program storage, they're all here. So what does this mean? It means that years from now, when other drum machines are sitting in the closet gathering dust, your TR-909 will still be on the job. Hook up the TR-909 through MIDI to a personal computer (like the Apple II or IBM PC). Only Roland has the Hardware and the Software to make it possible. The Most Expression. Compare the results you get from the TR-909 Rhythm Composer with any drum machine. Because why would you want a machine, when you can have a Rhythm Composer? RolandCorp US, 7200 Dominion Circle, LA, CA 90040.

Roland

RolandMakesItHappen!

Different Drummer

We don't call the TR-909 a drum machine for some very good reasons. True, it's a machine that makes drum sounds, but that's the end of any similarities between run-of-the-mill drum machines and the TR-909 Rhythm Composer. In fact, playing with the TR-909 is more like playing with a real drummer than anything else. Here's why. The Best Sounds. We start with digital recordings of real drums, then through a 3-D waveform analysis, re-create the sounds through a hybird digital/analog process. Not only does this provide the best drum sounds, but also the most flexible. Change the snap of the snare, the decay of the bass, you call it. The sounds you get are the sounds you really want. Even better-in addition to the 11 internal drum sounds, add up to 16 more drum sounds (digital and analog) through external sound modules. That means 27 drum sounds with no major surgery. The Best Programming.

@rowdyrabouw

Program a roll on most drum machines and you'll see why they're called machines. That's why the TR-909 gives you the choice of Step Programming (highly visual and accurate) PLUS the additional spontaneity of Real-time Programming. The TR-909 also gives the most expressive and easily programmed dynamics. **The Most Flexibility.** Think of any way to interface, and you'll find it on the TR-909. MIDI, Sync-24, Tape Memory Save/Load, RAM-Pak Program storage, they're all here. So what does this mean? It means that years from now, when other drum machines are sitting in the closet gathering dust, your TR-909 will still be on the job. Hook up the TR-909 through MIDI to a personal computer (like the Apple II or IBM PC). Only Roland has the Hardware and the Software to make it possible. **The Most Expression.** Compare the results you get from the TR-909 Rhythm Composer with any drum machine. Because why would you want a machine, when you can have a Rhythm Composer?

Roland Makes It Happen!

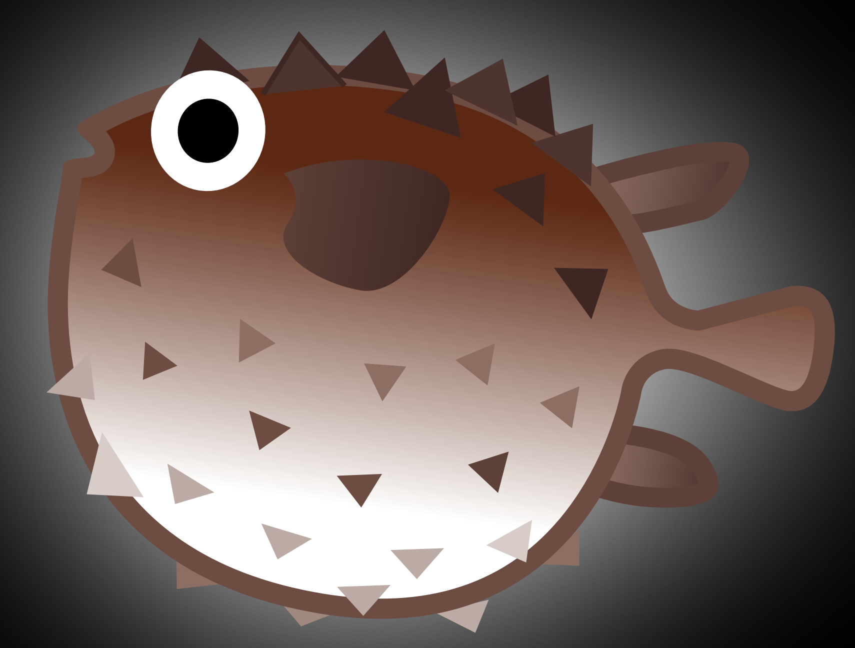
@rowdyrabouw

Program a roll on most drum machines and you'll see why they're called machines. That's why the TR-909 gives you the choice of Step Programming (highly visual and accurate) PLUS the additional spontaneity of Real-time Programming. The TR-909 also gives the most expressive and easily programmed dynamics. The Most Flexibility. Think of any way to interface, and you'll find it on the TR-909. MIDI, Sync-24, Tape Memory Save/Load, RAM-Pak Program storage, they're all here. So what does this mean? **It means that years from now, when other drum machines are sitting in the closet gathering dust, your TR-909 will still be on the job.** Hook up the TR-909 through MIDI to a personal computer (like the Apple II or IBM PC). Only Roland has the Hardware and the Software to make it possible. The Most Expression. Compare the results you get from the TR-909 Rhythm Composer with any drum machine. Because why would you want a machine, when you can have a Rhythm Composer?

Roland Makes It Happen!

@rowdyrabouw

BRITISH



ROWDY RABOUW

Used to be a Club DJ in the late 80's and early 90's

Build my first website in 1996

Front-End Focused Senior DevOps Engineer

Google Developer Expert in Web Technologies

WEB AUDIO API

The Web Audio API is a JavaScript API that provides a set of audio-related functionalities for creating and manipulating audio content in web applications.

AUDIOCONTEXT

The AudioContext represents an audio-processing graph containing audio sources, nodes, and audio destinations.

```
const ctx = new AudioContext();
```

OSCILLATORNODE

The OscillatorNode generates periodic waveforms for creating sounds in real-time.

```
const ctx = new AudioContext();
const osc = new OscillatorNode(ctx, {


});
```

FREQUENCY

The frequency property of the OscillatorNode determines the pitch of the generated sound. It represents the number of cycles per second, measured in Hertz (Hz).

```
const ctx = new AudioContext();
const osc = new OscillatorNode(ctx, {
  frequency: 440,

});
```

WAVEFORM

The OscillatorNode can generate different types of waveforms.
Each waveform has a distinct timbre and harmonic content.

SINE

TRIANGLE

SQUARE

SAWTOOTH

```
const ctx = new AudioContext();
const osc = new OscillatorNode(ctx, {
  frequency: 440,
  type: "sine",
});
```

```
const ctx = new AudioContext();
const osc = new OscillatorNode(ctx, {
  frequency: 440,
  type: "sine",
});

osc.connect(ctx.destination);
osc.start(ctx.currentTime);
osc.stop(ctx.currentTime + 2);
```

Roland
TR-909
RHYTHM COMPOSER

WAV (WAVEFORM AUDIO FILE FORMAT)

A WAV file is used for storing uncompressed audio data, resulting in high-quality audio reproduction.
It was developed by Microsoft and IBM in 1991.

```html
<button class="pad" data-note="40" data-wav="909-kick"></button>
<button class="pad" data-note="41" data-wav="909-clap"></button>
<button class="pad" data-note="42" data-wav="909-closed-hat"></button>
<button class="pad" data-note="43" data-wav="909-open-hat"></button>
```

```
<button class="pad" data-note="16" data-wav="909-kick"></button>
<button class="pad" data-note="41" data-wav="909-clap"></button>
<button class="pad" data-note="42" data-wav="909-closed-hat"></button>
<button class="pad" data-note="43" data-wav="909-open-hat"></button>

const pads = document.querySelectorAll(".pad");
```

```
<button class="pad" data-note="16" data-wav="909-kick"></button>
<button class="pad" data-note="41" data-wav="909-clap"></button>
<button class="pad" data-note="42" data-wav="909-closed-hat"></button>
<button class="pad" data-note="43" data-wav="909-open-hat"></button>

const pads = document.querySelectorAll(".pad");
pads.forEach((pad) => {




});
```

```
<button class="pad" data-note="16" data-wav="909-kick"></button>
<button class="pad" data-note="41" data-wav="909-clap"></button>
<button class="pad" data-note="42" data-wav="909-closed-hat"></button>
<button class="pad" data-note="43" data-wav="909-open-hat"></button>

const pads = document.querySelectorAll(".pad");
pads.forEach((pad) => {
  pad.addEventListener("click",       () => {


  });
});
```

```
<button class="pad" data-note="16" data-wav="909-kick"></button>
<button class="pad" data-note="41" data-wav="909-clap"></button>
<button class="pad" data-note="42" data-wav="909-closed-hat"></button>
<button class="pad" data-note="43" data-wav="909-open-hat"></button>

const pads = document.querySelectorAll(".pad");
pads.forEach((pad) => {
  pad.addEventListener("click", async () => {
    const sample = await loadFile(`${pad.dataset.wav}.wav`);

  });
});
```

```
<button class="pad" data-note="16" data-wav="909-kick"></button>
<button class="pad" data-note="41" data-wav="909-clap"></button>
<button class="pad" data-note="42" data-wav="909-closed-hat"></button>
<button class="pad" data-note="43" data-wav="909-open-hat"></button>

const pads = document.querySelectorAll(".pad");
pads.forEach((pad) => {
  pad.addEventListener("click", async () => {
    const sample = await loadFile(`${pad.dataset.wav}.wav`);
    playWav(sample);
  });
});
```

```
const loadFile =        (wav) => {


};

const playWav = (audioBuffer) => {


};
```

```
const loadFile = async (wav) => {
  const response = await fetch(wav);



};


const playWav = (audioBuffer) => {



};
```

```
const loadFile = async (wav) ⇒ {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();


};


const playWav = (audioBuffer) ⇒ {



};
```

```
const loadFile = async (wav) ⇒ {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await ctx.decodeAudioData(arrayBuffer);

};


const playWav = (audioBuffer) ⇒ {



};
```

```
const loadFile = async (wav) ⇒ {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await ctx.decodeAudioData(arrayBuffer);
  return audioBuffer;
};


const playWav = (audioBuffer) ⇒ {




};
```

```
const loadFile = async (wav) ⇒ {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await ctx.decodeAudioData(arrayBuffer);
  return audioBuffer;
};


const playWav = (audioBuffer) ⇒ {



};
```

```
const loadFile = async (wav) ⇒ {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await ctx.decodeAudioData(arrayBuffer);
  return audioBuffer;
};


const playWav = (audioBuffer) ⇒ {
  const wav = new AudioBufferSourceNode(ctx, { buffer: audioBuffer });


};
```

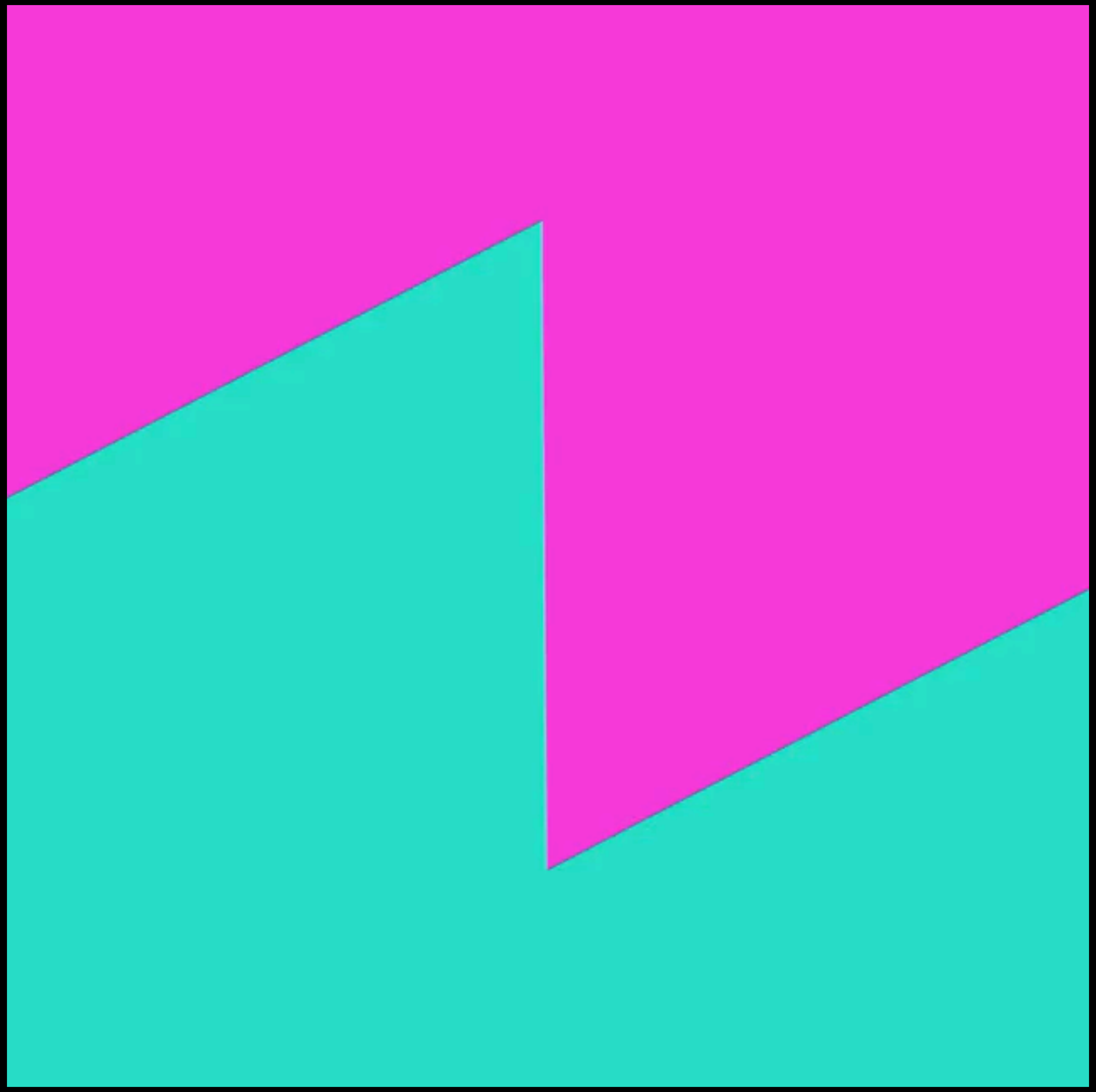
```
const loadFile = async (wav) => {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await ctx.decodeAudioData(arrayBuffer);
  return audioBuffer;
};


const playWav = (audioBuffer) => {
  const wav = new AudioBufferSourceNode(ctx, { buffer: audioBuffer });
  wav.connect(ctx.destination);

};
```

```
const loadFile = async (wav) ⇒ {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await ctx.decodeAudioData(arrayBuffer);
  return audioBuffer;
};


const playWav = (audioBuffer) ⇒ {
  const wav = new AudioBufferSourceNode(ctx, { buffer: audioBuffer });
  wav.connect(ctx.destination);
  wav.start();
};
```



TONE.JS

Tone.js is an open-source JavaScript library that provides a framework for creating interactive and dynamic music and sound in web applications.

```
const kick = new Tone.Player('909-kick.wav').toDestination();
const kick = new Tone.Player('909-kick.wav').toDestination();

kick.start();
```

```
<button class="pad" data-note="40" data-wav="909-kick"></button>
<button class="pad" data-note="41" data-wav="909-clap"></button>
<button class="pad" data-note="42" data-wav="909-closed-hat"></button>
<button class="pad" data-note="43" data-wav="909-open-hat"></button>

const pads = document.querySelectorAll(".pad");
pads.forEach((pad) => {
  pad.addEventListener("click", async () => {
    const sample = await loadFile(`${pad.dataset.wav}.wav`);
    playWav(sample);
  });
});
```

```javascript
const loadFile = async (wav) => {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await ctx.decodeAudioData(arrayBuffer);
  return audioBuffer;
};


const playWav = (audioBuffer) => {
  const wav = new AudioBufferSourceNode(ctx, { buffer: audioBuffer });
  wav.connect(ctx.destination);
  wav.start();
};
```

```
<button class="pad" data-note="40"></button>
<button class="pad" data-note="41"></button>
<button class="pad" data-note="42"></button>
<button class="pad" data-note="43"></button>
```

```html
<button class="pad" data-note="40"></button>
<button class="pad" data-note="41"></button>
<button class="pad" data-note="42"></button>
<button class="pad" data-note="43"></button>
```

```javascript
const kick = new Tone.Player('909-kick.wav').toDestination();
const clap = new Tone.Player('909-clap.wav').toDestination();
const closed = new Tone.Player('909-closed-hat.wav').toDestination();
const open = new Tone.Player('909-open-hat.wav').toDestination();
```

```
<button class="pad" data-note="40"></button>
<button class="pad" data-note="41"></button>
<button class="pad" data-note="42"></button>
<button class="pad" data-note="43"></button>


const kick = new Tone.Player('909-kick.wav').toDestination();
const clap = new Tone.Player('909-clap.wav').toDestination();
const closed = new Tone.Player('909-closed-hat.wav').toDestination();
const open = new Tone.Player('909-open-hat.wav').toDestination();

const pads = document.querySelectorAll(".pad");
pads.forEach((pad) => {
  pad.addEventListener("click", async () => {
    switch (pad.dataset.note) {
      case "40": kick.start();   break;
      case "41": clap.start();   break;
      case "42": closed.start(); break;
      case "43": open.start();   break;
    }
  });
});
```

@rowdyrabouw

WEB MIDI API

The Web MIDI API is a JavaScript API that allows web applications to communicate and interact with MIDI (Musical Instrument Digital Interface) devices connected to a user's computer or device.

AKAI professional
LPD8
LAPTOP PAD CONTROLLER
PAD 1
PAD 2
PAD 3
PAD 4
PAD 5
PAD 6
PAD 7
PAD 8
PROG 1
PROG 2
PROG 3
PROG 4
PROGRAM
PAD
CC
PROG CHNG
K1
K2
K3
K4
K5
K6
K7
K8
MIN
MAX

```
if (navigator.requestMIDIAccess) {
  navigator.requestMIDIAccess().then(succes, failure);
}
```

```
if (navigator.requestMIDIAccess) {
  navigator.requestMIDIAccess().then(succes, failure);
}

const succes = (midiAccess) ⇒ {







}
```

```
if (navigator.requestMIDIAccess) {
  navigator.requestMIDIAccess().then(succes, failure);
}

const succes = (midiAccess) => {

  const inputs = midiAccess.inputs;




}
```

```
if (navigator.requestMIDIAccess) {
  navigator.requestMIDIAccess().then(succes, failure);
}

const succes = (midiAccess) => {

  const inputs = midiAccess.inputs;

  inputs.forEach((input) => {
   input.addEventListener("midimessage", handleInput);
  });

}
```

$$\underset{\text{command}}{144}, \overset{\text{note}}{61}, \underset{\text{velocity}}{102}$$

```html
<button class="pad" data-note="40"></button>
<button class="pad" data-note="41"></button>
<button class="pad" data-note="42"></button>
<button class="pad" data-note="43"></button>
```

```
<button class="pad" data-note="40"></button>
<button class="pad" data-note="41"></button>
<button class="pad" data-note="42"></button>
<button class="pad" data-note="43"></button>

const handleInput = (input) => {









};
```

```
<button class="pad" data-note="40"></button>
<button class="pad" data-note="41"></button>
<button class="pad" data-note="42"></button>
<button class="pad" data-note="43"></button>

const handleInput = (input) => {

  const note = input.data[1];
  const velocity = input.data[2];





};
```

```
<button class="pad" data-note="40"></button>
<button class="pad" data-note="41"></button>
<button class="pad" data-note="42"></button>
<button class="pad" data-note="43"></button>

const handleInput = (input) => {

  const note = input.data[1];
  const velocity = input.data[2];

  if(velocity > 0) {
    const pad = document.querySelector(`[data-note="${note}"]`);
    pad.click();
  }

};
```

Roland
TR-909
RHYTHM COMPOSER

SEQUENCER

A sequencer is a device or software application used in music production to create, edit, and arrange musical sequences or patterns. It is a fundamental tool in electronic music production.

FIRE
GRID
SELECT
BROWSER
Analog Bass
PATTERN
CHANNEL
MIXER
USER 1
USER 2
MUTE
SOLO
STEP
ACCENT
NOTE
SNAP
DRUM
TAP
PERFORM
OVERVIEW
SHIFT
ALT
PATTERN/SONG
METRONOME
WAIT
COUNTDOWN
LOOP REC
AKAI PROFESSIONAL
FL STUDIO
AKAI PROFESSIONAL

SEQUENCER
pan low
pan high
pitch
bpm
120 bpm
Init
Midi
hbpm

```
<div class="kick">
  <label><input type="checkbox" data-mute="kick" /> </label>
  <label><input type="checkbox" data-note="54" /> </label>
  <label><input type="checkbox" data-note="55" /> </label>
  <label><input type="checkbox" data-note="56" /> </label>
  <label><input type="checkbox" data-note="57" /> </label>
  <label><input type="checkbox" data-note="58" /> </label>
  <label><input type="checkbox" data-note="59" /> </label>
  <label><input type="checkbox" data-note="60" /> </label>
  <label><input type="checkbox" data-note="61" /> </label>
  <label><input type="checkbox" data-note="62" /> </label>
  <label><input type="checkbox" data-note="63" /> </label>
  <label><input type="checkbox" data-note="64" /> </label>
  <label><input type="checkbox" data-note="65" /> </label>
  <label><input type="checkbox" data-note="66" /> </label>
  <label><input type="checkbox" data-note="67" /> </label>
  <label><input type="checkbox" data-note="68" /> </label>
  <label><input type="checkbox" data-note="69" /> </label>
</div>
```

```
const handleInput = (input) => {

  const note = input[1];
  const velocity = input[2];

  if (velocity > 0) {
    if (note >= 54 && note <= 117) {
      document.querySelector(`[data-note="${note}"]`).click();
    }
  }

}
```

```
const initSequencer = () ⇒ {

  Tone.Transport.scheduleRepeat((time) ⇒ {
    repeat(time);
  }, 16);

}
```

```
let index = 0;
const steps = 16;

const repeat = (time) => {

 // simplified, only showing kicks

  let step = index % 16;

  const muteKicks = document.querySelector('.kick [data-mute]');
  const kicks =
    document.querySelector(`.kick label:nth-child(${step + 2}) input`);

  if (!muteKicks.checked && kicks.checked) {
    kick.start(time);
  }

  index++;
}
```

```
const recorder = new Tone.Recorder();
const recorder = new Tone.Recorder();

const record = async () => {
  await recorder.start();
}
```

```
const recorder = new Tone.Recorder();

const record = async () ⇒ {
  await recorder.start();
}

const stop = async () ⇒ {
  const recording = await recorder.stop();










}
```

```
const recorder = new Tone.Recorder();

const record = async () ⇒ {
  await recorder.start();
}

const stop = async () ⇒ {
  const recording = await recorder.stop();

  const url = URL.createObjectURL(recording);
  const audio = document.createElement('audio');
  audio.controls = true;
  audio.src = url;


}
```

```
const recorder = new Tone.Recorder();

const record = async () ⇒ {
  await recorder.start();
}

const stop = async () ⇒ {
  const recording = await recorder.stop();

  const url = URL.createObjectURL(recording);
  const audio = document.createElement('audio');
  audio.controls = true;
  audio.src = url;

  document.querySelector('#recordings').append(audio);
}
```

FIRE
GRID
SELECT
BROWSER
Analog Bass
PATTERN
CHANNEL
MIXER
USER 1
USER 2
MUTE
SOLO
STEP
ACCENT
NOTE
SNAP
DRUM
TAP
PERFORM
OVERVIEW
SHIFT
ALT
PATTERN/SONG
METRONOME
WAIT
COUNTDOWN
LOOP REC
AKAI PROFESSIONAL
FL STUDIO
AKAI PROFESSIONAL

AKAI professional
LPD8
LAPTOP PAD CONTROLLER
PROGRAM
PAD
CC
PROG CHNG
PAD 1
PAD 2
PAD 3
PAD 4
PAD 5
PAD 6
PAD 7
PAD 8
PROG 1
PROG 2
PROG 3
PROG 4
K1
K2
K3
K4
K5
K6
K7
K8
MIN
MAX

WEB BLUETOOTH API

The Web Bluetooth API allows web applications to interact with Bluetooth devices. It provides a way for websites to discover nearby devices, establish connections with them, and exchange data.

```
const device = await navigator.bluetooth.requestDevice({
  filters: [
    { namePrefix: 'Bluetooth-Device' } // which device?
  ],
  optionalServices: [0xffe5] // what service(s)? → uuid
});
```

```
const device = await navigator.bluetooth.requestDevice({
  filters: [
    { namePrefix: 'Bluetooth-Device' } // which device?
  ],
  optionalServices: [0xffe5] // what service(s)? → uuid
});

// ‘array of objects’
const server = await device.gatt.connect();
const device = await navigator.bluetooth.requestDevice({
  filters: [
    { namePrefix: 'Bluetooth-Device' } // which device?
  ],
  optionalServices: [0xffe5] // what service(s)? → uuid
});

// ‘array of objects’
const server = await device.gatt.connect();

// ‘object’
const service = await server.getPrimaryService(0xffe5);
const device = await navigator.bluetooth.requestDevice({
  filters: [
    { namePrefix: 'Bluetooth-Device' } // which device?
  ],
  optionalServices: [0xffe5] // what service(s)? → uuid
});

// ‘array of objects’
const server = await device.gatt.connect();

// ‘object’
const service = await server.getPrimaryService(0xffe5);

// ‘property’
const characteristic = await service.getCharacteristic(0xffe9);
const device = await navigator.bluetooth.requestDevice({
  filters: [
    { namePrefix: 'Bluetooth-Device' } // which device?
  ],
  optionalServices: [0xffe5] // what service(s)? → uuid
});

// ‘array of objects’
const server = await device.gatt.connect();

// ‘object’
const service = await server.getPrimaryService(0xffe5);

// ‘property’
const characteristic = await service.getCharacteristic(0xffe9);

await characteristic.startNotifications();
```

POLAR

```
characteristic.addEventListener("characteristicvaluechanged", (event) ⇒ {


});
```

```
characteristic.addEventListener("characteristicvaluechanged", (event) ⇒ {
  const hbpm = event.target.value.getUint8(1);


});
```

```
characteristic.addEventListener("characteristicvaluechanged", (event) ⇒ {
  const hbpm = event.target.value.getUint8(1);
  Tone.Transport.bpm.value = hbpm;
});
```

DANK JE WEL!

rowdy.codes/ff